AFRL-SR-AR-TR-04-

0339

REPORT DOCUMENTATION PAGE

1. REPORT DATE *(DD-MM-YYYY)*
June 17, 2004

2. REPORT TYPE
Final Report -STTR

3. DATES COVERED *(From - To)*
8-15-03 to 5-14-04

4. TITLE AND SUBTITLE
Distributed Streams-based Data-Mining for Application Intrusion Detection

5a. CONTRACT NUMBER
F49620-03-C-0057

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Yurica, Kevin
Pande, Rahul
Motwani, Rajeev

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

RealTime Methods, Inc.
3490 Freedom Circle
Santa Clara, CA 95054

Stanford University
Computer Science Dept.
Gates Computer Science Bldg.
Stanford, CA 94305

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. Robert Herklotz
AFOSR / NM
4015 Wilson Blvd. Rm. 713
Arlington, VA 22203-1954

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR / NM

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This investigation considered the challenge of real-time, distributed data mining across high-level TCP/IP protocols for application layer intrusion detection. The approach taken was to focus on the fundamental challenges of; a) evaluating similarities between different application-level TCP/IP protocols, b) node-based header evaluation methods for HTTP, c) a communication strategy to support aggregation and coordination. This streams-based approach to real-time data mining appears to be a useful in many areas including; security monitoring, intrusion detection and sensor networks.

15. SUBJECT TERMS
Data mining, Real-time processing, Application security, Distributed processing

16. SECURITY CLASSIFICATION OF: UNCLASSIFIED			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE		30	19b. TELEPHONE NUMBER *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

20040706 004

Distributed Streams-based Data Mining for Application Intrusion Detection

Phase I Final Report

Version 1.0

ABSTRACT: This investigation considered the challenge of real-time, distributed data mining across high-level TCP/IP protocols for application layer intrusion detection. The approach taken was to focus on the fundamental challenges of; a) evaluating similarities between different application-level TCP/IP protocols, b) node-based header evaluation methods for HTTP, c) a communication strategy to support aggregation and coordination. This streams-based approach to real-time data mining appears to be a useful in many areas including; security monitoring, intrusion detection and sensor networks.

KEYWORDS: Data mining, Real-time processing, Application security, Distributed processing

COMPANY: RealTime Methods, Inc.

RESEARCH INSTITUTION: Stanford University

TOPIC: Storage Efficient Data Mining - AF SB032-037

1.0 Executive Summary

This investigation considered the challenge of real-time, distributed data mining for application layer intrusion detection across high-level TCP/IP protocols. In focusing on the application level, it's possible to view activities in real-time that may provide specific insight into human or machine behavior patterns. The rapid assessment of such behavioral patterns may reveal much about the health and status of applications, individuals, nodes and network services. The general approach taken was to focus on three fundamental challenges; a) characterizing the application layer challenge b) developing streams-based data evaluation methods, b) identifying a communication strategy that enables aggregation and coordination.

The data mining process developed evaluates an HTTP header stream by filtering and analyzing requests for particular conditions of interest. When the evaluation pipeline detects an anomaly of some type (i.e. - a URI parameter is outside an allowable range), the condition generates a notification event. Much of the analysis performed exploits the native structure and artifacts of the HTTP protocol. The communication headers captured, essentially create a 'bread crumb trail' that provides for step-by-step evaluation of user interactions. Discerning 'normal' behavior from 'abnormal' behavior is accomplished by working with resource and parameter values as well as expected paths through resource hierarchies. This exploitation of protocol-specific convention appears to create a suitable starting point for the further application of statistical measures.

The second major aspect- the communication strategy, was addressed using a commercial Web services platform as the basic communication infrastructure. While Web services infrastructure promises to fulfill the needs of message-oriented applications in the long-run, a couple of technology issues still exist. Today, Web services infrastructure is still largely oriented to simple data exchange such as passing parameter values or arrays, when both ends of the conversation understand the full context of the data being passed in advance. Ultimately, it is believed that Web services will work very well as a message-oriented communication infrastructure for applications similar to the one proposed here. However today, the commercially available infrastructure is only starting to address requirements for improved message-oriented functionality.

The analysis infrastructure investigated here appears to be well-matched to distributed processing scenarios where application monitoring routines need to be lightweight and storage is bounded. Applications where this approach may be especially productive include; security monitoring, telecommunications, ad hoc wireless networks, and sensor networks.

2.0 Scientific/Technical Objectives

This research investigated the potential for streams-based data mining to deliver improved functionality in the area of application-level intrusion detection.
The primary technical challenges to be investigated were as follows:

1. Investigate the feasibility of streams-based data mining in a distributed environment
2. Investigate the effectiveness of cluster-based analysis methods for application intrusion detection.
3. Identify appropriate control, messaging and reporting mechanisms to support streams-based data mining and intrusion detection.

3.0 Approach Taken

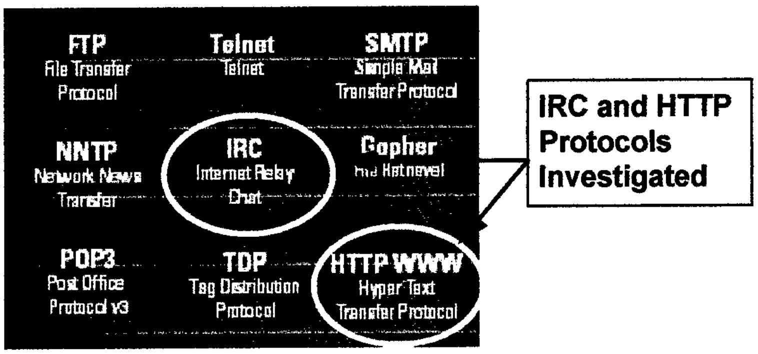
The general approach taken was to define, model and simulate the services required in a prototype system to evaluate feasibility. The technical investigation consisted of the following five activities.

1. Investigate HTTP and IRC communication headers as a basis for application layer intrusion detection
2. Evaluate HTTP header analysis alternatives and algorithms
3. Design and develop a lightweight data mining process to extract information from HTTP communication headers
4. Define control, communication and reporting mechanisms
5. Simulate system behavior using component models

3.1 The Extraction of Communication Headers

The target data source for this investigation was high-level TCP/IP-based protocols. Considering the OSI communication model, this effort focuses on capturing information at the Application Layer (Layer 7), which is responsible for providing appropriate semantics and meaning to the transported data. In contrast, most packet-level firewalls and IDS systems are focused on the Transport Level (Layer 4), where messages are segmented into packets. The most popular high-level TCP/IP protocols include; FTP, HTTP, NNTP, SMTP, POP3, IRC and others.

Figure 1 - Application Layer Protocols based on TCP/IP
(Excerpt from Agilent Protocol Map)



For the investigation of node-based data evaluation methods, two application layer protocols in particular were looked at closely- IRC and HTTP. The differences between HTTP and IRC were thought to be substantial enough that each would require a distinct effort. HTTP was selected as the best candidate for an initial implementation effort since the IRC protocol allows for dynamic indirection- potentially removing the 'conversation' from the view of the original monitoring node.

The general data stream parameters of interest in this investigation can be divided into three groups: 1) connection parameters, 2) communication headers, 3) content parameters or features.

The premise behind the focus on communication headers was that the extraction of this data may provide an efficient 'view' into the context of a communication exchange. It is worth challenging this notion however. Protocol-specific header data has largely not been used in modern IDS systems which tend to rely more on packet inspection methods. Clearly if you need to monitor every port and every packet, the use of a proxy server is not a good choice. However, in the domain of application security, the examination of communication headers to assess the health of an application is not uncommon.

Much of the data that is intercepted here in the form of HTTP communication headers is also available in Web server logs. Along these lines, many of the same questions that someone might try to answer through the process of 'Web mining' on log files can start to be answered in near real-time using the techniques developed here. In the domain of user behavior analysis and clickpath analysis, the extraction of communication headers essentially provides a step-by-step sequence of nearly every action a user takes.

While there appear to be advantages to the use of communication headers, there are also disadvantages. With respect to the overall generality of this approach, it

may be regarded as 'somewhat limited' since the headers and protocol artifacts exploited here for HTTP can not be generally applied to other TCP/IP protocols such as; IRC, NNTP, SMTP, etc.

The advantages of the use of communication header methods relative to packet inspection include:

- Produces a precise account of a communication exchange with context and semantics
- Sessions can be actively intercepted or halted (pass/no pass)
- Encrypted sessions can be decrypted at the proxy intercept point

The disadvantages of the use of communication headers relative to packet inspection include:

- Not as general as packet-level inspection
- Requires a proxy server for the protocol of interest
- The use of a proxy server has the potential to create client-server compatibility issues

As more and more enterprise content streams are encrypted, the use of a proxy server remains one of the few good alternatives to decode encrypted sessions and gain insight into encrypted stream activity. Accordingly, some leading network security solutions include both firewalls and proxies within a single product (including the Microsoft 2004 Internet Security and Acceleration Server used here). This integration of firewall and proxy server functionality offers a high level of flexibility to help address a wide variety of potential deployment scenarios.

3.2 What Questions can be answered through Header Analysis?

In the domain of communication analysis, it's relatively easy to entertain interesting 'what if?' questions. However, getting answers to such questions on a real-time basis is computationally expensive so one must carefully select each analysis task to fit within a performance budget that is limited. Along these lines, any practical analysis method selected for real-time use must be evaluated from a 'cost-benefit' perspective. Analysis methods that can be applied generally to the broadest range of scenarios, become highly desirable from the perspective that 'lightweight and efficient' are probably key to success.

While a large number of different questions can potentially be answered through the analysis of communication headers, one of the advantages of listening to the TCP/IP stack at the application layer is the ability to capture the *context* or *semantics* of an exchange. Some the most valuable questions that might be answered here are already being answered today- except in less expedient

ways. Today, firewalls and directory servers are setup to enforce broad policies on resource access. However, more specific detail indicating 'who' accessed 'what' and 'when' is typically only available from server-based log files which often represent the only detailed transaction record in an organization.

The investigation focused on addressing the basic questions about users such as; *who? what?* and *when?* Getting answers to these questions requires more specific analysis focused in the following areas.

- Request origination - Where did the request come from? What domain, IP, or organization? Is this origination known?
- Resource requested - What resource is requested? What URI or particular name-value pairs were called? Is this resource known?
- Request sequence - Extraction of session information, links or paths between resources or the lack thereof. Is this sequence known?

3.3 Practical Methods for Communication Header Analysis

A number of different analysis methods were considered to determine which might be successfully applied to the task of HTTP communication header analysis. The data mining and analysis methods investigated included; string categorization, range checking, clustering, and Bloom filters. The critical factor in designing streaming algorithms is reducing computational workload and managing scaling factors- which ideally should have linear characteristics. While many different statistical measures could be applied here, at a practical level, we can afford only a modest workload and still keep up with real-time arrival rates. One intuitive concept described in work at Stanford is that good streaming algorithms can often be derived from conventional analysis algorithms[1].

Table 1 – Summary of Analysis Methods Investigated

Summary of Analysis Methods Investigated			
Analysis Method	**Application Examples**	**Strengths**	**Weaknesses**
String Categorization	+ Origination domain, and IP, destination domain and IP, others + URI resource, name-value pairs, others	+ Simple + General + Efficient	+ Difficult to adapt strings to most numerically oriented statistical methods
Range Checking	+ URI name-value pair checking + Origination or destination domain and IP	+ Can be adapted to work with strings + Can answer general 'set membership' questions	+ Requires application-specific information to treat strings numerically in most cases
Clustering	+ Name-value pairs, feature vectors, others + Strong opportunity to pre-calculate values for use in r/t analysis	+ Very efficient way to identify 2nd order relationships	+ Computationally expensive to perform all work in real-time
Bloom Filters	+ URI resource and name-value pairs	+ Highly efficient way to differentiate 'seen' vs. 'never before seen' requests	+ Requires data cleaning and alignment + Implementation more difficult in the case of dynamic content

Since header informàtion is initially gathered as strings, string-oriented evaluation methods are the easiest to apply. Certain header fields such as the URI field, potentially contain many embedded parameters of interest. In the case of the URI field, the machine, domain, resource identifier, and a number of name-value pairs may be present. With respect to application intrusion detection and clickpath analysis, such details are both important and interesting. However, for any analysis process to successfully use such embedded parameters, one must first successfully extract, filter and align them.

A lot of research has been done in the area of Web Usage Mining which directly or indirectly addresses some of the issues involved in the extraction of; navigational patterns, ordering relationships, clustering of web usage sessions, and prediction of behavior. Most commonly this work is focused on Web logs and possibly supplemented with the analysis of web content or other information. "One fundamental challenge in all these problems is that the raw data is in the form of sequences, and not vectors. As a result, it is difficult to apply many well developed data mining techniques for vector type data."[2]

3.3.1 String Categorization

String-based analysis methods can be applied to the exploitation of protocol-specific header data and artifacts. The resource requested, parameters passed and clickpath may all be reflective of the behavior of the user. To efficiently work with such sequences of events, mechanisms are required which quickly categorize large numbers of requests and efficiently determine their nature.

String categorization can be used to extract, record and catalog recurring string sequences by capturing the relationships between substring segments. Strings can be parsed into a series of sub-strings which are then ordered and arranged into a hierarchy that reflects the characteristics of the passing content.
One way to make string manipulations more efficient is to tokenize recurring substrings into a common form that normalizes variants. In the case of HTTP, URIs can be tokenized into a series of resource parameters and often a series of name-value pairs as well.

In the following URIs, the domain, resources and name-value pairs have been extracted and organized as sub-strings that can then be classified in a hierarchy.

URI #1: http://www.rtmethods.com/home/company/contact.html

Primary domain:	rtmethods.com
Sub-domain:	www
Resource token series:	home company contact.html

URI #2: http://ncid.rtmethods.com/cgi-bin/gx.cgi/applogic+ftcontentserver?pagename=sitea/tsearch/common&z5=1&z4=3&t3=1051026&z2=10001

Primary domain:	rtmethods.com
Sub-domain:	ncid
Resource token series:	cgi-bin

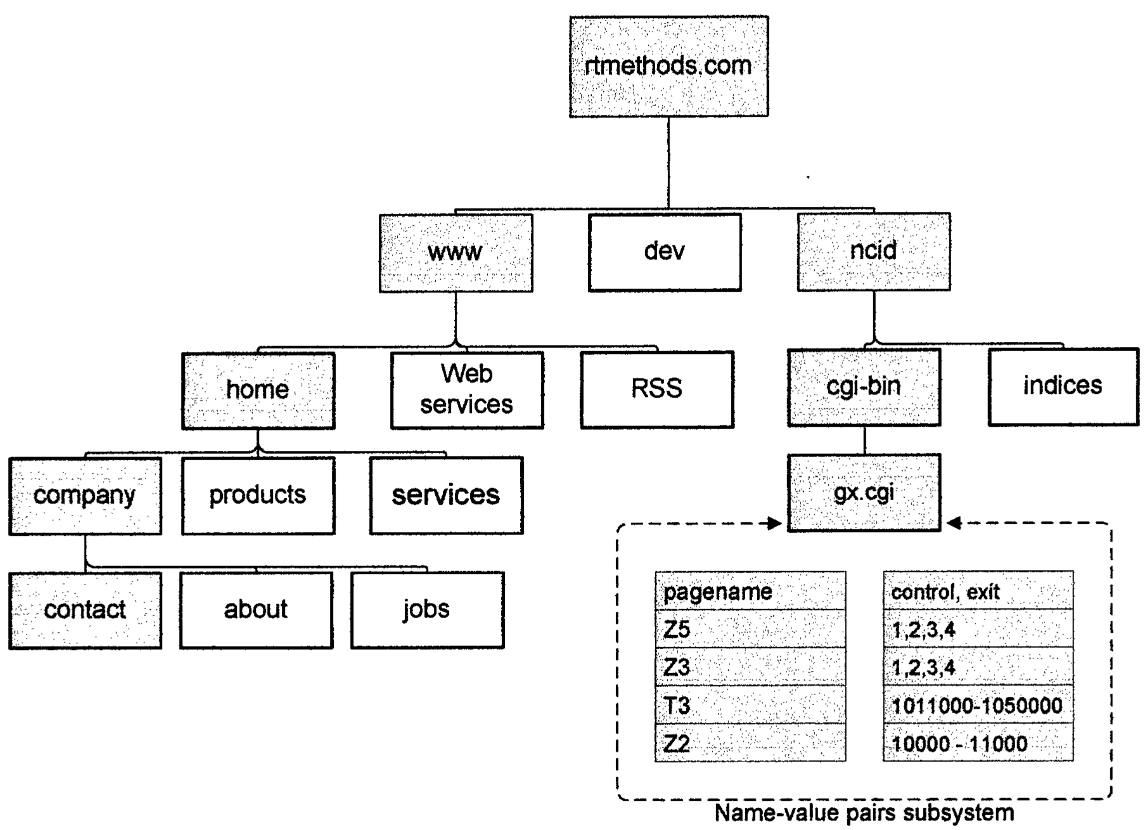
gx.cgi
applogic+ftcontentserver

Name-value pairs:

pagename	sitea/tsearch/common
z5	1
z4	3
t3	1051026
z2	10001

In each case, the primary domain is rtmethods.com, and we can map the relationship between these elements by constructing a simple tree as follows. The chain of relevant tokens for each of these URI examples is shown in gray in the content hierarchy below.

Figure 2 – Categorization Hierarchy Example



3.3.2 Range-Checking Methods

The use of range-checking appears promising in the domain of communication header analysis- in part, due to the inherent flexibility of range checking methods. While most easily applied to numbers, range checking variants can also be

applied to strings. In the domain of numerical sorting, significant value is associated with each ordered position relative to the decimal point (or least significant integer placeholder). With character-based sorting, each character is simply compared against others in the same position starting from the left. In the case of name-value pairs, a character-based sort can be applied to string-based parameters. While a string may not be 'in range' numerically, when it's bounded by two other string values, it is certainly characterized within certain limits. This type of string-boundary characterization is relatively effective and efficient.

One of reason that range-checking appears to be so useful in the real-time processing domain is simplicity. It is very easy to specify a particular range-based test condition and also very easy to describe a 'pass' or 'failure' condition. Second, range checking appears to be very efficient from a computational workload perspective. Third, range-checking is inherently flexible and can be applied to many things. For example, an IP number can be range checked to determine its relationship to other known subnets.

The simplicity of range-checking also seems to offer advantages in the domain of distributed applications. Since specific values are enumerated and tested, it is relatively straightforward to share this type of information with other nodes. Distributed range-checking schemes such as Top-k, offer the opportunity to extend range checking evaluation into the distributed fabric of the network.

3.3.3 Clustering Analysis Methods

The application of many common clustering analysis methods to HTTP header analysis is challenging for two reasons. First, header data is generally captured as strings which makes numerical comparisons difficult. While there is a wealth of header information available, a significant amount of cleaning and alignment work is required before second order relationships can be reliably extracted.

Second, while there are certainly a large number of conceptual opportunities for the application of clustering, practical considerations make many applications difficult. From a 'workload' perspective, many common clustering methods are computationally expensive and difficult to fit into a real-time evaluation scheme. One way to bring the benefits of clustering into a real-time evaluation environment is to pre-calculate clustering-based comparison values. In the case where comparison data can be analyzed or optimized off-line, a very significant design win is available from an efficiency standpoint.

An interesting opportunity to apply this hybrid clustering approach is found in the area of clickpath analysis methods. "Data mining from web access logs is a process consisting of three consecutive steps: data gathering and pre-processing for filtering and formatting the log entries, pattern discovery which consists of the

use of a variety of algorithms such as association rule mining, sequential pattern analysis, clustering and classification on the transformed data in order to discover relevant and potentially useful patterns, and finally, pattern analysis during which the user retrieves and interprets the patterns discovered."[3]

Some off-line methods for clickpath analysis rely on clustering methods to evaluate page 'similarity' with other pages. One such approach explored here involves the use of a 'similarity' measure that is calculated based on the content features of a Web page or resource. If the similarity/dissimilarity measure for different content resources is pre-calculated, then the real-time evaluation process can be as simple as comparing requests against known values.

3.3.3.1 *Pre-calculated Path Clustering for Session Analysis*

To address the challenge of session analysis, an approach that relies on pre-calculated path clustering was investigated. The general idea is to use off-line similarity clustering to determine most likely path vectors through a resource hierarchy. The results of this similarity measure are calculated off-line and converted to an index which is associated with each defined node in a resource hierarchy. Two simple variants of this method were noted; a) using path tokens to assess similarity, b) using link analysis to assess similarity.

In the case of path tokens, similarity is computed base on the position of a resource within a content hierarchy. Resources that are adjacent to one another are considered similar, while those that are distant are deemed dissimilar. The process of calculating path token values proposed here is adapted from similar work performed by Banerjee and Ghosh[4]. In their work, they described the process of calculating path tokens as follows:

"This method first determines the intersection of any two paths under consideration as defined by the longest common subsequence (LCS) between the two sequences of pages visited by each user. Then, a similarity value is computed over this LCS as a function of the closeness in the times spent on the corresponding pages weighted by a certain importance factor. Using these similarity values, a graph is constructed whose nodes are the paths."[5]

For any particular pair of sequences, as the *path* corresponding to the session under consideration, we are interested in finding the maximum-length or longest common subsequence (LCS) given two paths or sequences of page-visits. To calculate the LCS for a pair of sequences, the following procedure suits our purposes.

Step1: Compare each corresponding token of the two token strings one by one from the beginning, until the first pair of different tokens is encountered.

Step2: Compute the similarity of two resources by first determining the length of the longest token string among the two. Then give a weight to each level of tokens: the last level of the longest token string is given weight 1, the second to the last is given weight 2, etc. Next, the similarity between two token strings is defined as the sum of the weight of those matching tokens divided by the sum of the total weights.

In the case of link analysis, the number of ways that two different resources are linked or associated by hyperlinks can be measured and indexed using a Web robot. This is similar to the link analysis approach used by Google which evaluates the 'authority' and 'popularity' of different references. For purposes here, once hyperlinks are extracted other content features can generally be ignored. We consider that two different resources have 'high' similarity when they include many of the same links or are pointed to a similar set of references.

If the two pages are totally different, i.e. no corresponding hyperlinks, their similarity is 0.0. If the two pages are exactly the same, their similarity would be 1.0- perfect matching between two session sequences. The optimal match is the alignment with the highest score reflecting the best fit between a series of requests.

3.3.4 Bloom Filters

The final method that we consider for communication header analysis was the use of Bloom filters[6]. Bloom filters are compact data structures for answering set-membership type queries with a high degree of precision. Given a list of strings and a key string, a bloom filter will answer whether the key string is contained within the list with a low probability of error. The filters work by representing each stored list element as a low dimension vector of bits. Query strings are then converted to like bit sequences and are compared to the stored bits in an efficient manner to determine set membership. The representation of each string by only a few bits results in the filters being extremely fast and economical; however, the drawback of this technique is that it introduces some lossiness into the data and results in a small false positive rate i.e. occasionally a bloom filter will say an element is in the filter even though it is not. It is, however, possible to parameterize the filters so that this error rate is made arbitrarily small with a corresponding incremental increase in the filter's space requirement.

The filters were first described in a 1970 paper by Burton Bloom[7], and have since found use in password appropriateness verification[8], unix spell checkers, ice-berg queries, packet routing, IP tracing along with a myriad of other applications in databases and networks[9]. In fact, nowadays the filters are considered to be so widely applicable that some senior scientists from IBM and Harvard have coined the Bloom filter principle: "Wherever a list or set is used, and space is a consideration, a Bloom filter should be considered. When using a Bloom filter, consider the potential effects of false positives."[10]

The use of Bloom filters offers a very efficient way to distinguish between 'never-seen-before' and 'seen-before' requests. Specifically, the use of Bloom filters to ascertain whether the parameters of an incoming HTTP request are 'in range' or 'not' was considered. This proposed approach involved storing all in-range strings for a given request parameter in a Bloom filter and then querying the filter to check whether the value of the corresponding parameter in an incoming HTTP request was in the stored list or not. An affirmative reply for all parameters in incoming request would imply that that HTTP request was (with high probability) in range and had been 'seen-before', a negative reply for even one parameter in the incoming request would mean that that request had 'never-been-seen-before'. The advantage that of using bloom filters to support this type of analysis, was that the filters require far less storage space than the other approaches (raw data, hashing, patricia trie's etc), but still are very accurate.

To test the suitability of this approach, a bloom filter implementation for the SQL Server environment was constructed with an expected false positive rate of 0.15 % (i.e. we would expected the filter to misclassify 2 strings for every 10,000 strings queried against it.) We stored 5000 strings of a fixed length in the filter, and then queried this structure with 100,000 random strings (of identical length as the strings stored in the filter). After each test we tabulated the size required by the bloom filter and the actual misclassification rate observed. The results are presented as follows.

Table 2 – Bloom Filter Accuracy Trials for Various String Lengths

String length	Size of raw data	Size of bloom filter	Theoretical misclassification rate	Actual misclassification rate
3	104 KB	40 KB	0.15%	4.1%
5	120 KB	40 KB	0.15%	0.16%
10	200 KB	40 KB	0.15%	0.16%
20	200 KB	40 KB	0.15%	0.15%
50	352 KB	40 KB	0.15%	0.15%
100	608 KB	40 KB	0.15%	0.15%
200	1112 KB	40 KB	0.15%	0.15%

In the results above, the 'size of the bloom filter' column refers to the size of the table (in SQL Server) required to store the bloom filter containing the 5000 strings. The 'size of the raw data' column gives the size of the table (in SQL Server) required to store the same strings as raw text (i.e. not encoded in the bloom filter) and has been provided for the purpose of comparison.

The results indicate that, for our application, in all cases bloom filters use far less space than just storing the raw data (approx. 3 to 28 times less space) and are still very accurate. In most cases their precision is still close to the configured value of 99.85%, but for small strings (of length 3) this accuracy falls to about

95%. It should be noted that the increased error for small strings can easily be addressed in our implementation by appending to a small constant suffix to each stored string to ensure that all strings in the filter are of length larger than 5.

The above tests indicate that Bloom filters are appropriate for the set membership queries required by our application. However, to be applied in a practical way, data alignment and conditioning of header data is required. For example, name-valued pairs appended to a URL are valid when presented in any order. Prior to casting a Bloom filter, the order of name-value pairs needs to be normalized in some consistent way. Other important data conditioning steps include the cleaning of escape characters and other string artifacts which might create variability in the way 'equivalent' requests are presented.

3.4 The Lightweight Data Mining Process Developed

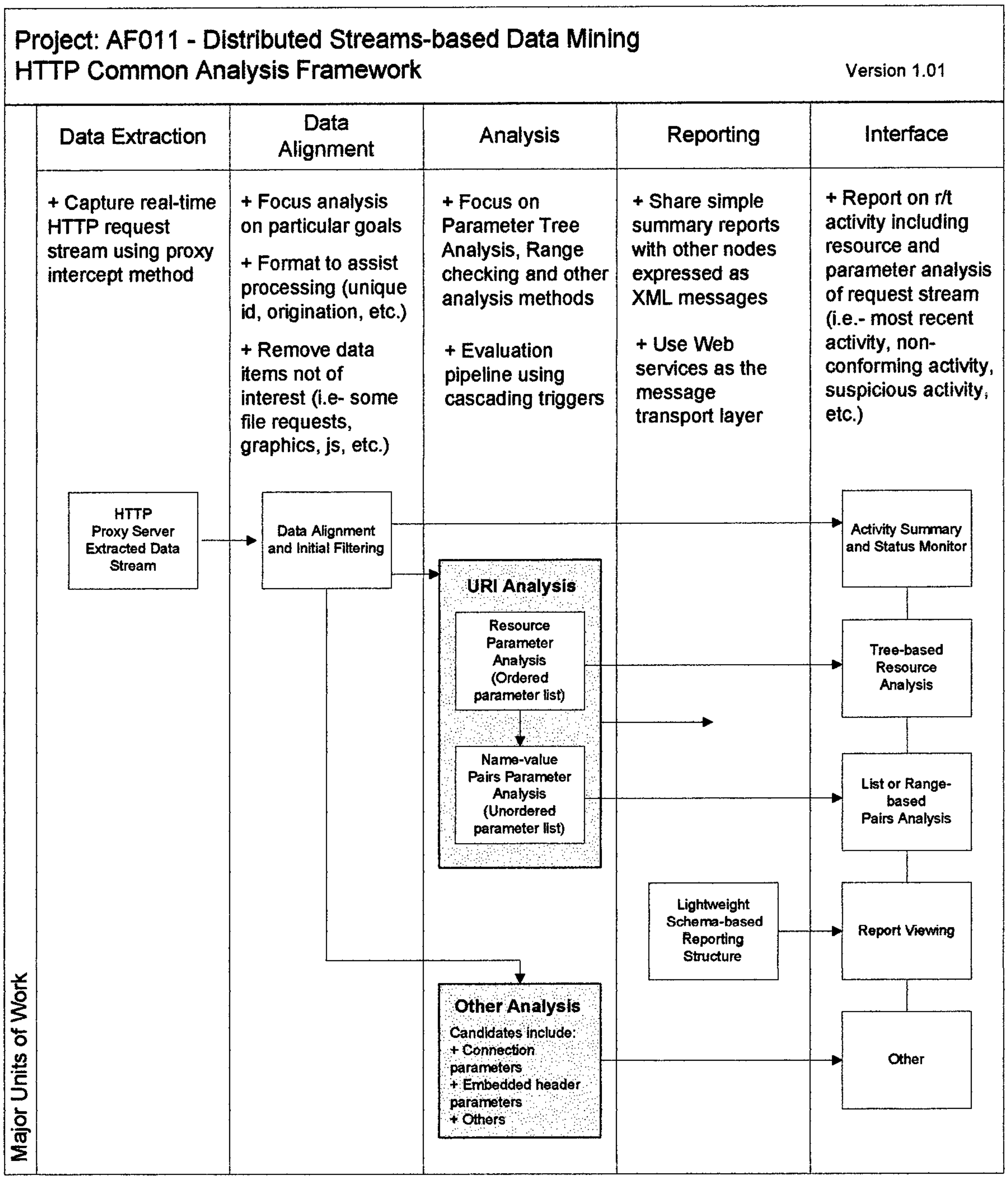
A lightweight data mining process was developed that filters, aligns and evaluates inbound requests against specified values and recent history. The communication headers captured here, essentially create a 'bread crumb trail' that provides for step-by-step evaluation of every resource a user requests. The high-level functionality implemented in this request evaluation pipeline can be summarized as follows.

- Distinguishes between 'seen before' versus 'never seen before' requests
- Hostname Matching - resolves synonyms, etc.
- Path Matching – resolves URI path elements
- Continuous Parameter Range Checking – range checks parameters in continuous ranges
- Discontinuous Parameter Range Checking – range checks parameters in discontinuous ranges
- Indexed-based evaluation for additional discriminators- such as pre-clustered path vectors based on URI analysis

This data mining process was implemented in the SQL programming language and runs on the Microsoft SQL Server 2000 relational database. While this same evaluation pipeline could be implemented outside of a database, the benefits to using a database include; rapid data capture, support for math and statistics, Web services connectivity, data integrity checking and more.

The data mining process developed can be expressed as a series of standard data mining 'steps' or procedures. Once data is extracted, it is cleaned and aligned in preparation for further analysis. The general flow of the data mining process can be seen in the diagram that follows.

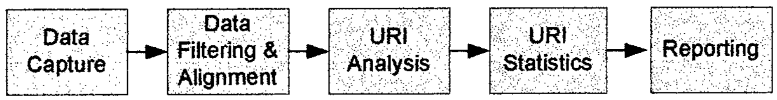
Figure 3 – Functional Diagram of HTTP Analysis Framework



At each stage in the processing pipeline, data items are processed by a series of stored procedures which are triggered when new data arrives. A Sliding Window implementation is used to define the working set at each stage in the data pipeline and automatically drops data items based on their age or conformity.

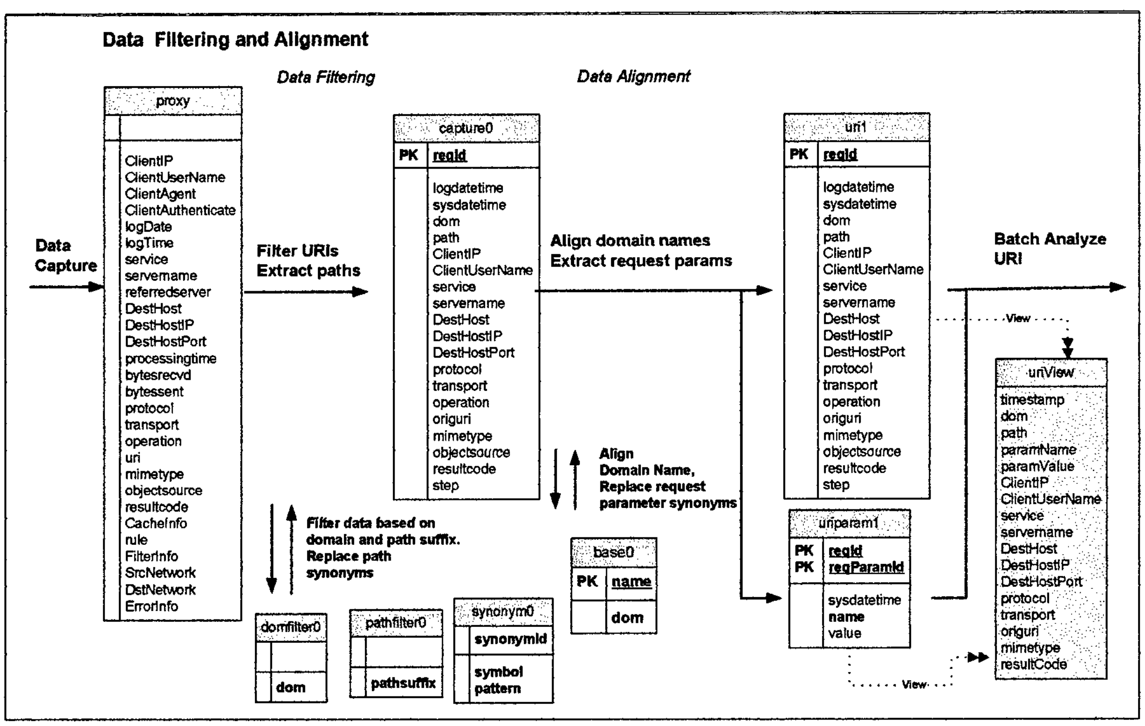
The data model discussion that follows is summarized at a high-level in the following sequence diagram. Data items enter the process at the capture point on the left and proceed step-by-step to the right.

Figure 4 – Data Mining Pipeline Stages



The Data Filtering and Alignment data model which follows illustrates initial data conditioning steps. First, data is captured by the proxy server and written in sequential blocks to the database. Second, a unique ID is assigned to each data item and preliminary filtering conditions are applied. Third, domain names are resolved against a synonym list which cross-references multiple IP addresses or domains which should be attributed to a single entity.

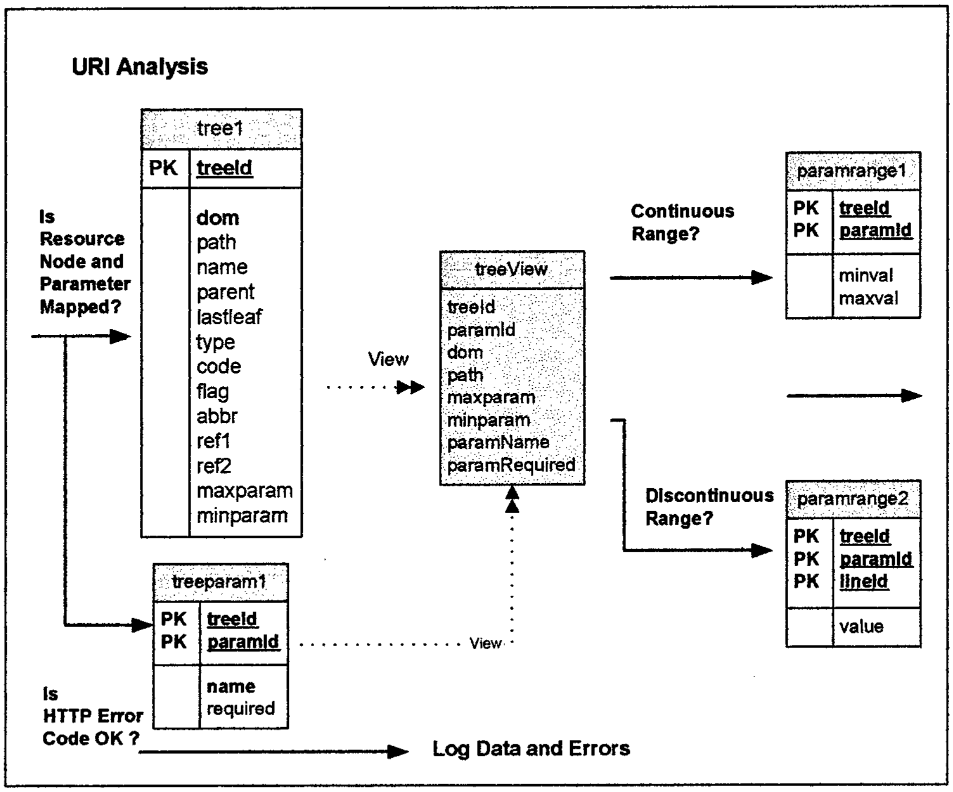
Figure 5 – Data Filtering and Alignment Unit



Once data items are conditioned and initial filtering is accomplished, data items are forwarded to the URI analysis unit which parses each URI string. While the implementation specifics of application URIs vary widely, there is consistent structure present which can be exploited. Static Web resources are typically arranged in a hierarchy that often mirrors the file system directory structure where

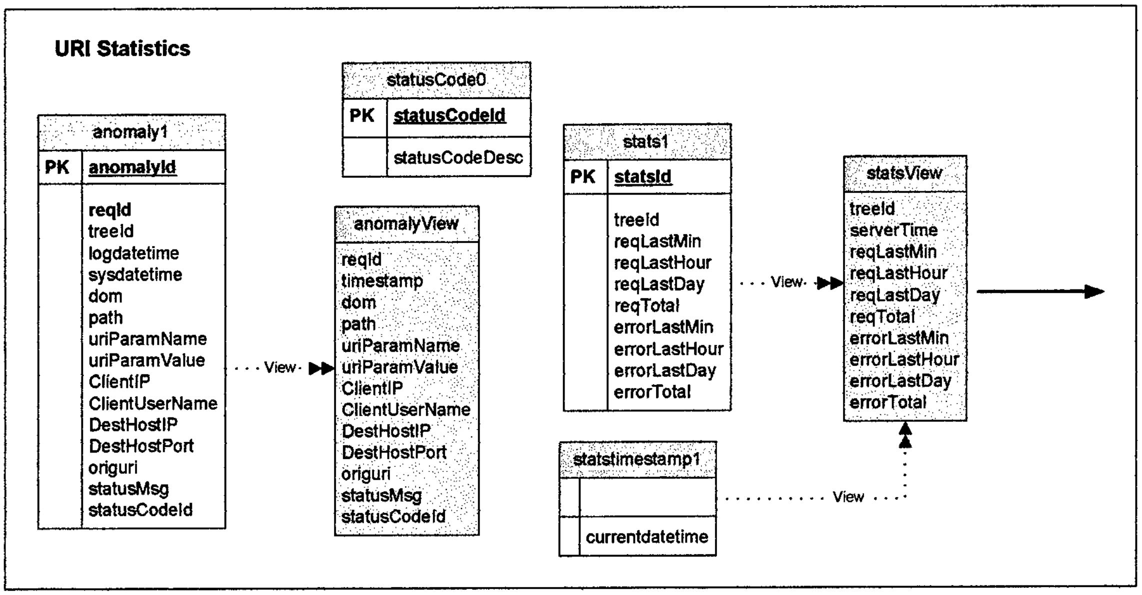
the content resides. And dynamic Web resources are typically exposed using a set of name-value pairs that represent the essential inputs to an information application. Of course, it is also common to have these conventions combined. The URI analysis unit addresses both of these situations and essentially characterizes every element of every request.

Figure 6 – URI Analysis Unit



In the case where a request has been 'seen before' and is deemed 'valid', the request is statistically recorded so that the occurrence is captured and characterized. In addition to simply counting requests, the URI Statistics Unit keeps a running count of activity levels over multiple Sliding Windows of time. Activity is recorded for the standard time windows of 'last minute', 'last hour' and 'last day'. Sliding Window evaluation parameters can be set for any time period of interest and for any number of independent instances.

Figure 7 – URI Statistics Unit



3.5 Communication and Reporting

A publication-subscription model was chosen that supports the exchange of light-weight event-oriented messages in an efficient manner. This model effectively removes communication burden from data producers and assigns it to a data broker that shares information across an 'n x n' matrix of nodes. The JBI Commercial Standards Working Group reviewed the use of Web services to address Pub-Sub scenarios in findings they reported in 2002.[11]

SOAP-based Web services offer promise as a ubiquitous machine-to-machine message transport mechanism. However, shortcomings in the use of Web services as a messaging fabric were identified. Today with Web services, there is no support for the notion of a general delivery 'destination'. However, there are workarounds that enable posting a message to a specific network end-point or machine. One new Web Services standard currently being pursued by the W3C may help change this; WS-MessageDelivery[12]. This effort promises to overlay an SMTP-like message delivery framework on top of SOAP. While similar to the previous WS-Addressing initiative, this new effort is thought to be more compatible with the existing WSDL standard.

In this investigation, a simple polling model was implemented using Web services, SOAP and XML. The test client subscribes to data from a remote evaluation node using Web services. Using the Microsoft .Net framework, the evaluation node is polled at regular intervals by the monitoring client to retrieve an activity summary. The .Net Web services implementation that was developed

for this application proved reliable down to polling intervals as small as 1/3 of second. At polling intervals of less than 1/3 second, update requests started to be dropped. To construct a more scalable architecture, two-way push-pull architecture is needed. Not only should updates be available via regular polling, but certain exceptions should be propagated to a client immediately. While there are some work-around approaches to accomplish this objective, pushing a message to a client on an exception basis would certainly be made easier with the implementation of a standard such as WS-MessageDelivery.

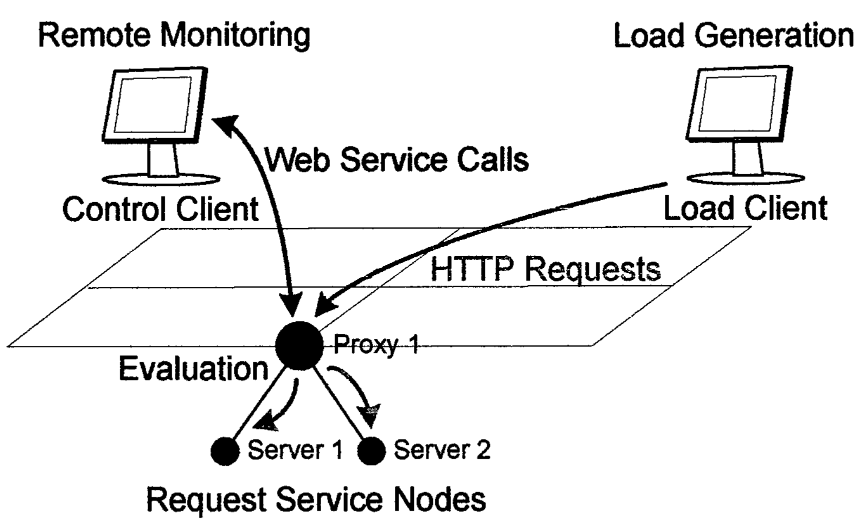
4.0 Performance Testing

The test case developed was based on a single evaluation node which was connected to a remote test client. The test client gathered the output of the data mining process from the remote evaluation node at regular intervals via Web service requests.

4.1 Description of Test Case

The goal of the performance testing was to assess the nature of the workload that could be accomplished. A desktop PC was configured as an evaluation node and was subjected to increasing numbers of requests until requests started to get dropped. The point at which application requests started to be dropped was designated as the maximum number of requests per second that the application could service on that specific machine.

Figure 8 – Diagram of Performance Testing Environment



4.2 Test System Configuration

The test system developed is based on the Microsoft Windows operating system and .Net Framework. Microsoft SQL Server was used as the database, however, other relational database products are also suitable for our purposes. Further details on the hardware and software used for testing purposes are described in the table below.

Table 3 –Performance Test System Configuration

Test System Configuration	
Hardware Configuration	**Software Configuration**
Intel 2.4 GHz Pentium 4 CPU	Microsoft Windows Server 2003
ASUS 9000 Motherboard	Microsoft Internet Security and Acceleration (ISA) Server 2004 Beta
512 MB RAM	Microsoft SQL Server 2000
100 MB Ethernet	

4.3 Testing Performed

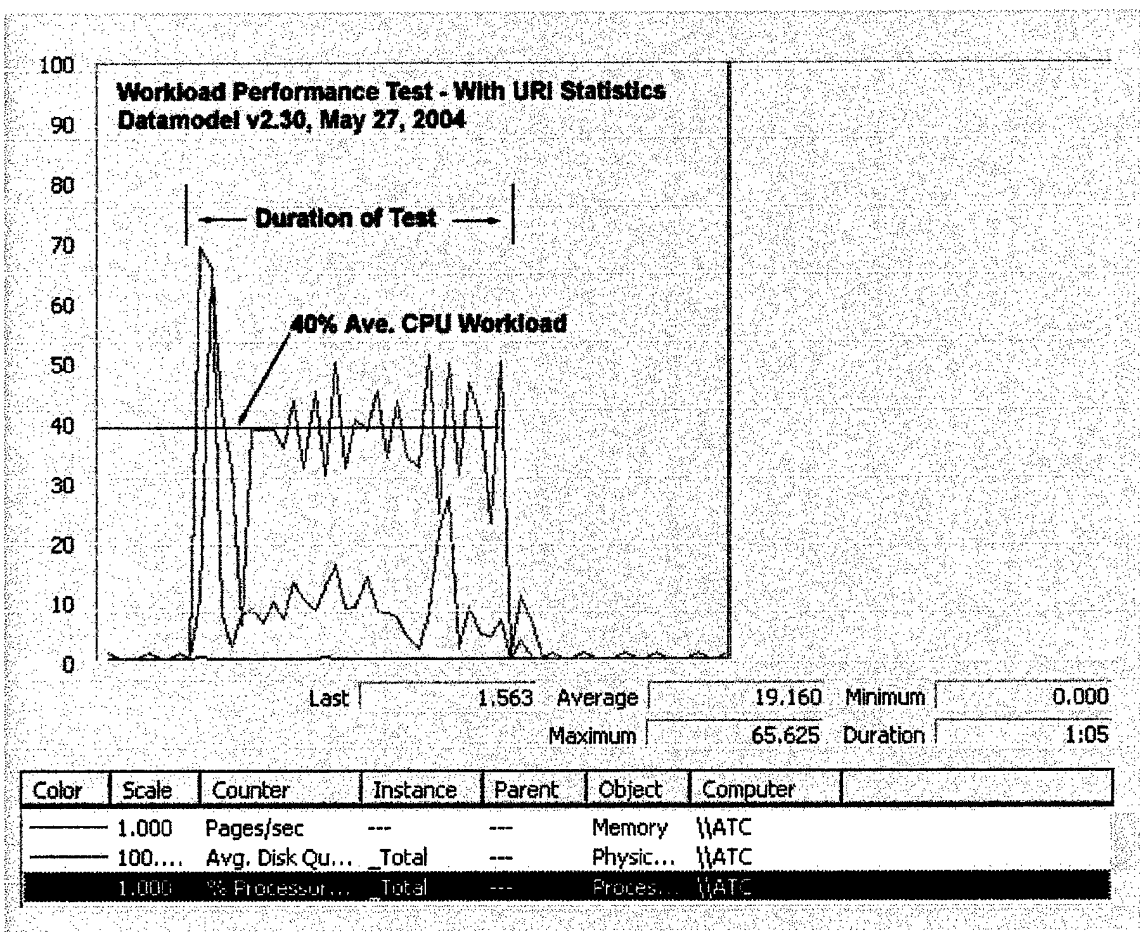
The test node was subjected to a request load that was generated across a LAN. A test client was developed in Java that would dispatch a specified number of HTTP requests each second where a specified fraction of the requests were either 'good' or 'bad'. The 'good' requests were already categorized in the URI Analysis tree and would pass the inspection portion of the evaluation process. The 'bad' requests had never been seen before and would result in anomalies being reported. In the case of 'bad' requests, anomaly reporting resulted in a slightly greater workload.

To assess the impact of incremental pipeline stages on the performance of the system overall- two different scenarios were run. In the first case, the complete evaluation pipeline was used and the second case, the URI Statistics Unit was turned off to reduce overall workload.

Case 1 – Performance Testing of Full Data Model

The test results shown below were recorded on May 27, 2004 using Datamodel Version 2.30 with the URI Statistics unit turned on. The workload applied was 25 HTTP requests per second issued from a test client over a 30 second period. An average CPU workload of 40% was observed. For this test system and configuration, an arrival rate of 25 requests per second may represent a practical limit. Subsequent trials at higher request arrival rates (30 and 35 req./sec.) both showed a significant percentage of dropped requests. During these subsequent trials at higher requests rates, CPU load tended to increase, however the total number of requests processed during the duration of test rarely exceeded the case when requests were arriving at a more modest rate and no requests were dropped.

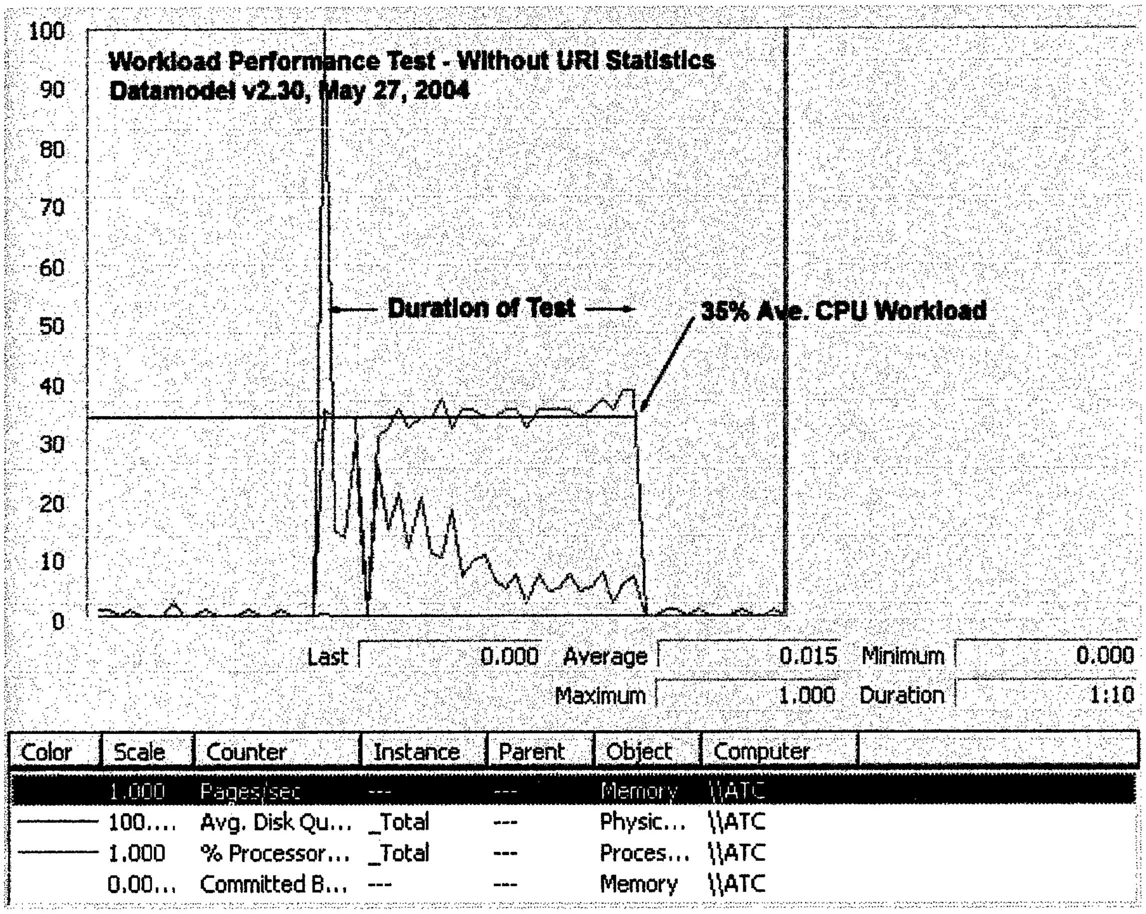
Figure 9 - Test Results for Full Data Model - Arrival rate of 25 requests/sec.



Case 2 – Performance Testing of Reduced Data Model

The test results shown below were recorded on May 27, 2004 using Datamodel Version 2.30 with the URI Statistics Unit turned off. The workload applied was 30 HTTP requests per second issued from a test client over a 30 second period. An average CPU workload of 35% was observed. At higher arrival rates (35 and 40 requests per second), some requests were dropped, suggesting that a practical limit had been reached around 30 requests per second.

Figure 10 - Test Results for Reduced Data Model (URI Statistics Unit Turned Off) - Arrival rate of 30 requests/sec.



Comparing the Case 1 and Case 2 test results between, we observed that the test system was able to keep up with a higher arrival rate when the analysis workload was simplified with the URI Statistics Unit turned off. This performance sensitivity to the depth of the analysis performed confirmed the 'workload vs. performance' balance that one might intuitively expect in this case.

It should be noted that while performance was a design goal, there are many 'known' optimization techniques that could be used to further increase efficiency. For example, our table-update mechanism relies on the use of triggers to detect the arrival of new data. Under high-volume conditions additional efficiency may be gained by 'batching' updates at specific increments of time.

4.4 General Discussion on Performance Results

The performance results obtained suggest that useable results for an application of this type are available today. From the 25 request per second load supported here on desktop PC, it is conceivable that a 3 to 4X improvement in performance is achievable through a combination of optimizations in the software and faster hardware. Performance on the order of 100 req./sec. would certainly be considered useable in most circumstances and might be sufficient to front-end a small server farm.

4.5 Data Storage Efficiency and Sliding Windows

Through a series of less formal tests, we were also able to observe that performance was directly impacted by the size of the Sliding Window used in certain table sets. When the Sliding Window size was increased and the core tables accumulated thousands of rows of data, a noticeable degradation in performance could be observed. Conceptually, this is expected since the number of data rows represents the 'working set' of the evaluation process. Simply put, the more data that needs to be 'handled', the greater the workload.

In the case of the data conditioning, alignment, and filtering steps of the evaluation process, there appears to be little advantage to maintaining any historical data in some cases. Accordingly, as soon as each of the conditioning, alignment, and filtering steps are completed, all data at that stage can be discarded. On the contrary, in the later stages of the evaluation process, there are benefits to storing data in the form of evaluation results. For example, in the model used here, data items that are not fully 'understood' are stored or reported as anomalies.

The Sliding Window model for data processing appears to be a natural fit for storage-efficient data mining. One of the advantages of a Sliding Window approach to a processing pipeline is that emphasis can be placed where ever it is desired. The performance characteristics of a particular implementation are easily tailored to suit specific objectives.

If we assume that the structured output of our HTTP evaluation process is a substitute for logging and storing every request received, then we can

calculate the storage efficiency of the model under varying conditions. It should be noted that the storage efficiency here is dependent on a variety of variables- including our ability to efficiently catalog the content or the request sources. The analysis being performed largely exploits similarity and redundancy. If every request received was 'out of the blue' and completely unique, our pipeline would record the request as an anomaly- and essentially log every request. The storage efficiency of our solution in this case would be no better than simply logging each request. On the other hand, as similarity increases across the pool of requests received, the storage efficiency of this model becomes more compelling. If the number of valid resource 'destinations' is limited and the overall percentage of 'valid' requests is high then the storage efficiency of our model can be quite high. In the table below, storage efficiency is calculated for a wide range of request and resource conditions.

Table 4 –Storage Efficiency under Various Conditions

Ave. Ratio of Requests to Unique URI Destinations	Percentage of Valid Requests	Storage Space Required Compared to Full Logging (1.0)	Storage Efficiency Compared to Full Logging (1.0)
1 to 1	100%	1.0	1.0
1 to 1	90%	1.0	1.0
1 to 1	80%	1.0	1.0
1 to 1	70%	1.0	1.0
10 to 1	100%	0.10	10
10 to 1	90%	0.11	9
10 to 1	80%	0.13	8
10 to 1	70%	0.14	7
100 to 1	100%	0.01	100
100 to 1	90%	0.011	90
100 to 1	80%	0.013	80
100 to 1	70%	0.014	70
1000 to 1	100%	0.001	1000
1000 to 1	90%	0.0011	900
1000 to 1	80%	0.0013	800
1000 to 1	70%	0.0014	700
10,000 to 1	100%	0.0001	10,000
10,000 to 1	90%	0.00011	9000
10,000 to 1	80%	0.00013	8000
10,000 to 1	70%	0.00014	7000

5.0 Positive and Negative Results Summaries

5.1 Positive Results Summary

1. Examined four possible methods for HTTP header analysis.

   - String-based categorization
   - Range checking
   - Clustering
   - Bloom filters

2. Implemented string-based data mining methods to examine HTTP header data.

   - Detects 'seen before' vs 'never seen before' requests
   - Parameter range checking (continuous and discontinuous)
   - Resolve synonyms and hostname matching
   - Computes request statistics for Sliding Window implementation
   - Most closely related to 'Web mining'

3. Established acceptable real-time performance in a working implementation. A 2.4 GHz, 512MB RAM desktop PC system intercepting in-bound requests supported a request arrival rate of 25 req./sec. It was also noted that workload is dependent on the depth of analysis performed. For example, a higher processing rate was observed (30 req./sec.) when the 'statistics' stage of the pipeline was turned off.

4. Demonstrated Web services-based communication architecture. Web services infrastructure holds promise as a message-oriented data exchange mechanism for distributed deployments. Web services are relatively simple to build and deploy. While current Web services infrastructure can be used to create additional message-oriented functions, custom messaging infrastructure provides little basis for widespread interoperability.

5. 2 Negative Results Summary

1. The initial direction of the investigation was overly optimistic with respect to how request parameters might be resolved in a numerically relevant manner. After examining many different clustering methods, only string or character-based clustering methods appear to be viable in the most general sense.

2. The task of building the HTTP evaluation pipeline was initially underestimated. The development of real-time multi-stage evaluation

models in a SQL programming environment proved more challenging than anticipated.

3. The use of Web services as a messaging architecture posed some challenges. While most of the required functionality is present in commercial Web services infrastructure, today, some key pieces related to 'addressing' and 'delivery' are still formative.

6.0 Conclusion

This investigation examined a number of different approaches to HTTP communication header analysis. In the end, the most productive analysis approaches seemed to be the simple ones. In the real-time domain, performance is nearly always an issue. The simpler and more efficient an evaluation method, generally the better. The use of string categorization, range checking and Bloom filters all appear to be techniques that are well suited to analysis goals here. The use of some clustering techniques also appears to be viable- especially those where some clustering measures can be pre-calculated to minimize runtime workload.

The Sliding Window model used here to evaluate streaming data appears to be a natural fit for storage-efficient data mining. One of the advantages of a Sliding Window approach to evaluation is that the size of the Sliding Window can be varied to place emphasis where ever it is needed.

While there are certainly many useful questions that can be answered using real-time evaluation methods, perhaps the most valuable questions in the near-term are the pragmatic ones- such as; who, what, where and when. When questions such as these are routinely addressed by a service-oriented architecture, we can begin to extract higher order patterns and relationships in real-time.

7.0 Commercialization Scenario

The data collection, integration and alignment tasks typically associated with log file analysis are sizeable tasks in many organizations- requiring storage, computing power, manpower and time to manage. One commercialization scenario currently being pursued is packaging the HTTP analysis pipeline, Web services implementation and client pieces as an add-on monitoring product for Microsoft's 2004 ISA Server. Based on our experience in developing with 2004 ISA Server, it appears to be a viable platform for more specialized analysis functions such as those described here.

8.0 Additional Resources

Additional resources are available from the project Web site:
http://www.rtmethods.com/research/af/
id: stanford pw: 94305

Further details are available on: Data models, Web services, Client software, IRC evaluation, and more.

9.0 References

[1] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. "First choose a linear time algorithm that performs well on static data. Repeatedly compose this favored algorithm in layers – each subsequent layer inputs the (weighted) cluster centers from the previous layer, and outputs O(k) clusters. The final layer ensures that only k clusters remain."

[2] Arindam Banerjee and Joydeep Ghosh, Clickstream Clustering using Weighted Longest Common Subsequences, http://www.lans.ece.utexas.edu/~abanerjee/papers/01/lcs.pdf

[3] Weinan Wang and Osmar R. Zaıane, Clustering Web Sessions by Sequence Alignment. http://www.cs.ualberta.ca/~zaiane/postscript/dexa2002.pdf

[4] Arindam Banerjee and Joydeep Ghosh, Clickstream Clustering using Weighted Longest Common Subsequences, http://www.lans.ece.utexas.edu/~abanerjee/papers/01/lcs.pdf

[5] Weinan Wang and Osmar R. Zaıane, Clustering Web Sessions by Sequence Alignment. http://www.cs.ualberta.ca/~zaiane/postscript/dexa2002.pdf

[6] B. Bloom. Space/time tradeoffs in in hash coding with allowable errors. CACM, 13(7):422-426, 1970.

[7] B. Bloom. Space/time tradeoffs in in hash coding with allowable errors. CACM, 13(7):422-426, 1970.

[8] Manber, U., Wu, S. An algorithm for approximate membership checking with application to password security, Information Processing Letters 50 (1994), 191-197

[9] Broder, A., Mitzenmacher, M. Network applications of Bloom filters: a survey, Allerton 2002

[10] Broder, A., Mitzenmacher, M. Network applications of Bloom filters: a survey, Allerton 2002

[11] Joint Battlespace Infosphere, Commercial Standards Paper, http://www.rl.af.mil/programs/jbi/documents/CommercialStandardsPointPaper.doc

[12] WS-MessageDelivery Version 1.0, http://www.w3.org/Submission/2004/SUBM-ws-messagedelivery-20040426/